NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

C437

HIGH SPEED OUTPUT INTERFACE FOR A MULTIFREQUENCY QUATERNARY PHASE SHIFT KEYING SIGNAL GENERATED ON AN INDUSTRY STANDARD COMPUTER

by

Robert Daniel Childs

December 1988

Thesis Advisor: P. H. Moose

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE		Form Approved OMB No 0704-0188
1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE		
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)		10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) HIGH SPEED OUTPUT INTERFACE FOR A MULTIFREQUENCY QUATERNARY PHASE SHIFT KEYING SIGNAL GENERATED ON AN INDUSTRY STANDARD COMPUTER

12 PERSONAL AUTHOR(S) CHILDS, Robert Daniel

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ________ TO ______	1988 December	62

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Communication; MFQPSK; Fourier

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A multiple frequency quaternary phase shift keyed signal is generated using a complex Fast Fourier Transform on an industry standard personal computer and is output using direct memory access through a digital to analog converter. The output is permitted at rates of up to the maximum direct memory access rate of the computer. An assembly language program loop, direct hardware output, and high level language output are compared as alternate solutions to the problem of outputting a data stream contained in the computer primary memory.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL MOOSE, P.	22b TELEPHONE (Include Area Code) (408)-646-2838	22c OFFICE SYMBOL 62Me

DD Form 1473, JUN 86 *Previous editions are obsolete* SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 UNCLASSIFIED

High Speed Output Interface for a Multifrequency Quaternary Phase Shift Keying Signal Generated on an Industry Standard Computer.

by

Robert Daniel Childs
Lieutenant, United States Navy
B.S.E.E., University of Washington, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1988

ABSTRACT

A multiple frequency quaternary phase shift keyed signal is generated using a complex Fast Fourier Transform on an industry standard personal computer and is output using direct memory access through a digital to analog converter. The output is permitted at rates of up to the maximum direct memory access rate of the computer. An assembly language program loop, direct hardware output, and high level language output are compared as alternate solutions to the problem of outputting a data stream contained in the computer primary memory.

C, I

DISCLAIMER

Some of the terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis are listed below following the name of the firm holding the trademark:

Advanced Micro Devices Co. ..	AMD
Borland International, Inc. .	Turbo Pascal, Turbo Pascal Numerical Methods Toolbox, Turbo Assembler, and the Turbo Debugger
Integrated Device Technology Inc.	IDT
Intel Corporation	Intel
Jameco Electronics	Jameco
Microsoft Corporation	Microsoft, MS-DOS
Zoran Corporation	Zoran

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logical errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Over the past several years a multiple frequency quaternary phase shift keyed (MFQPSK) communications system has been developed at the Naval Postgraduate School (Proctor,1985) (Gray,1986) (Whitacre,1986). Several implementations have been proposed which require specialized hardware construction with no flexible means of actually generating the signal instead requiring the signal to be generated offline. A simpler implementation housed in the industry standard computer was desired. The benefits of using a general purpose personal computer as the driver were anticipated to be that the transfer of data would be simplified if the output device was physically connected to the generation device and that the technique could be simply replicated by others desiring to generate such signals.

B. MFQPSK SIGNAL

The MFQPSK encoding is similar to that currently used by 1200 baud modems using the Bell 212A Standard **d**ifferential **p**hase **s**hift **k**eying (DPSK) in that data is represented by one of four phase shifted versions of a single frequency (tone) in each baud of a transmission. A baud is a discrete grouping of information, in this case, capable of representing two binary bits of information.

A four baud sequence of the four possible phase choices on a single tone is demonstrated in Figure 1 where the relative phase selections are +0 degrees, +180 degrees, +270 degrees, and +90 degrees. Note that the shift is referenced to the previous phase position, not the initial position of a +45 degree phase shifted sinusoid.

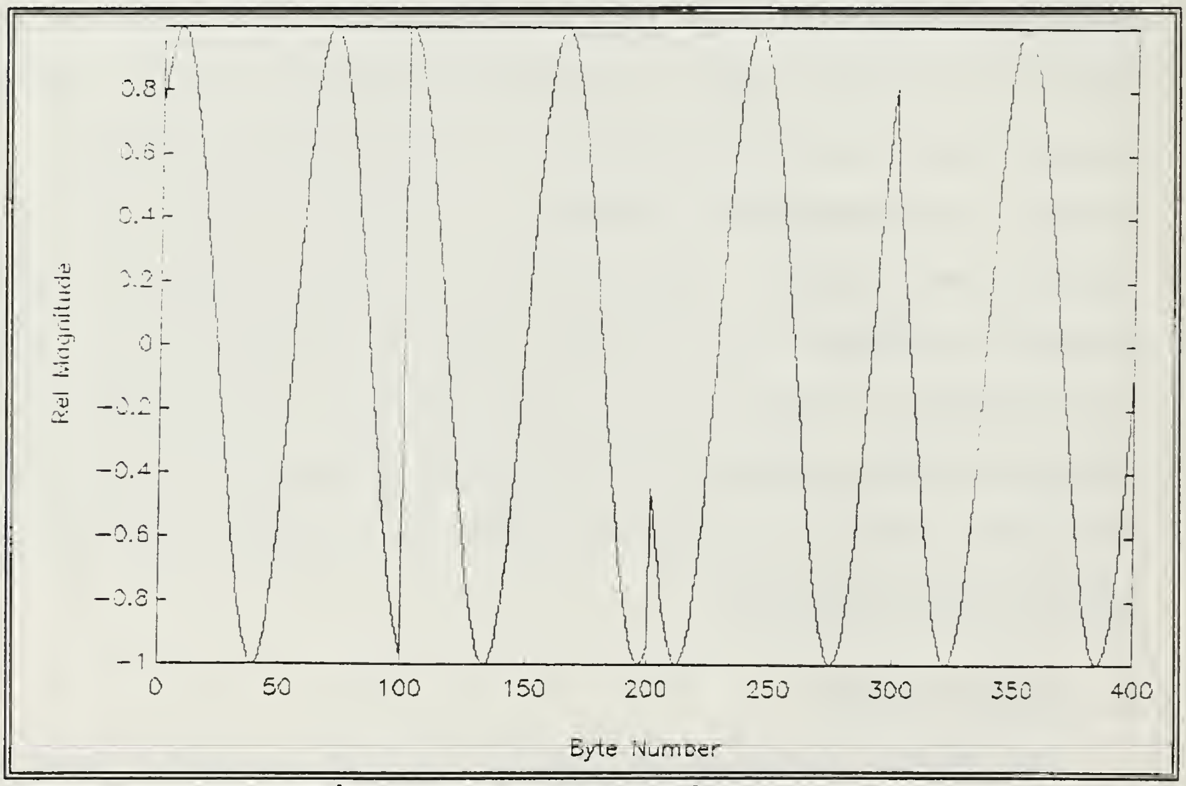


Figure 1. Four Baud Sequence

The modem technology does not directly shift the tone's phase but rather uses the algebraic addition and subtraction of a sine wave and cosine wave, the quadrature or orthogonal components, of the tone to modulate this phase information as is shown in Figure 2 which represents the third baud of the above sequence--a total of +135 degrees shift, or a tone shifted to the second quadrant.

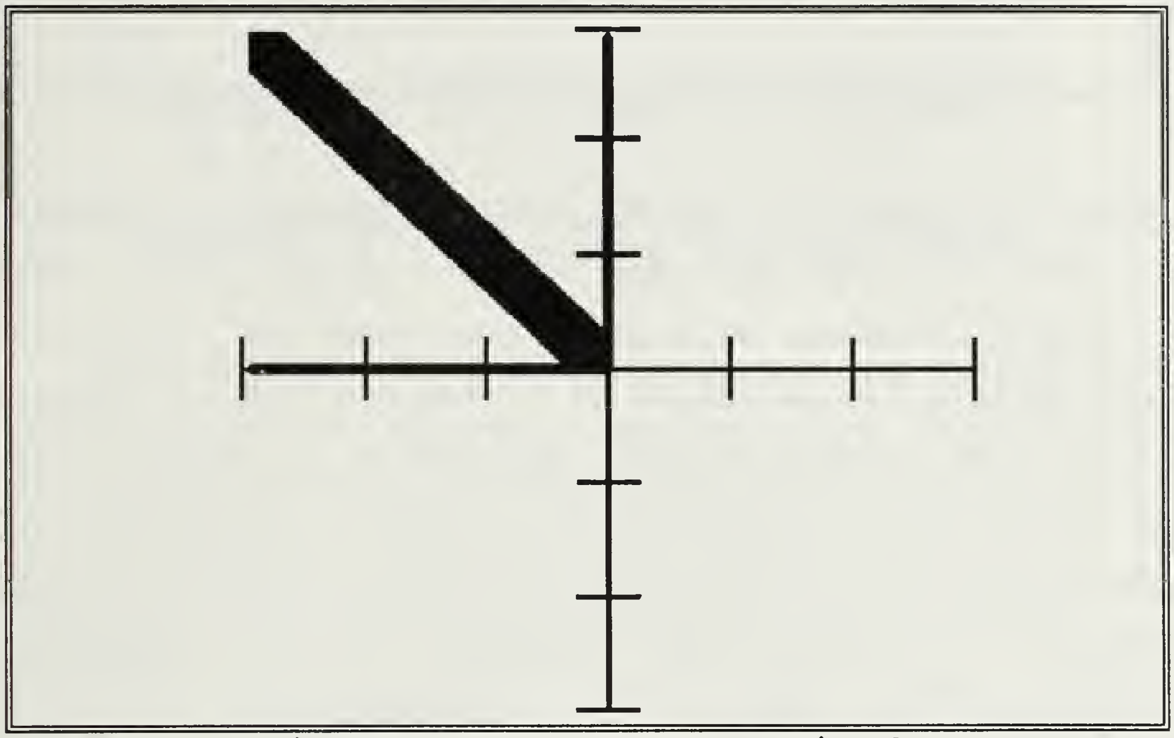


Figure 2. Second Quadrant Signal

While this method is employed reasonably well with one discrete frequency (or two for the case of a full duplex modem) it is not very practical for multiple frequencies. A method of modulating frequencies with phase information, which is suited better to many frequencies, is the frequency spectrum to time signal transformation of the **F**ast **F**ourier **T**ransform (FFT).

Figure 3 demonstrates this principle for a single frequency and a phase in the first quadrant (a phase shift of +90 degrees) of a 16 frequency bin system and introduces a new

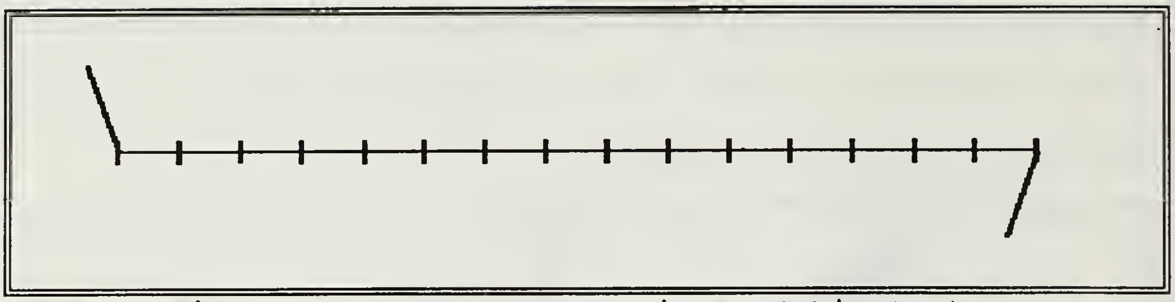


Figure 3. One Frequency in a 16 Bin System

graphical convention where the frequency spectrum is displayed showing both phase and magnitude on a single graph. Each frequency location is a polar plot with the line length representing the magnitude of a frequency and the angular displacement representing the phase shift. A vertical line would indicate a positive purely imaginary frequency. The FFT method is the approach taken to generate the MFQPSK signal. The phases are specified in their appropriate frequency bins which, in turn, predetermines, for real signals, the phases for the bins symmetrically located about the half sampling (Nyquist) frequency bin of the transform. The spacing of these frequency bins (or frequency resolution) is determined by dividing the Nyquist frequency by the number of bins on each side. Sampling time is the direct inverse of this frequency bin size. An example that encodes information in four frequencies per baud in a 16 frequency bin system is shown in Figure 4. If the resulting transformed signal has the sampled-and-held signal updated with a new value at a frequency of, for example, 60 Hz, the Nyquist frequency is 60

Hz, the top frequency bin represents 120 Hz and each of the frequency bins have a size of 7.5 Hz.

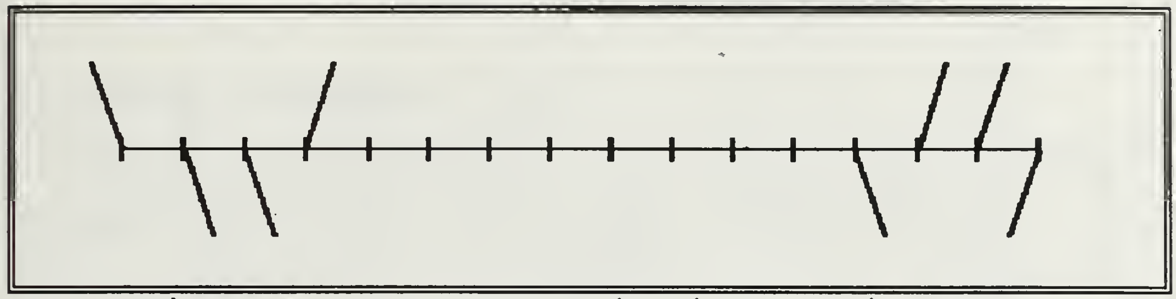


Figure 4. Four Frequencies in a 16 Bin System

Following the Inverse Discrete Fourier Transformation of such a signal, the completely real time domain signal, which consists of a series of 16 discrete steps made continuous by holding the sample until the next discrete step is due, is transmitted carrying phase information in each of the discrete frequencies simultaneously. The overall signal closely resembles bandlimited white noise. Figure 5 shows the time domain signal of a typical 256 tone, 4096 bin packet or baud.

C. OVERVIEW

The purpose of this study was to design, optimize, and construct a prototype output expansion board for the personal computer bus. The sister design problem of constructing an expansion board for the received signal is the subject of a concurrent thesis study, although to a large extent the problem is greatly simplified by choices made on the output problem. Chapter Two addresses the communication system as

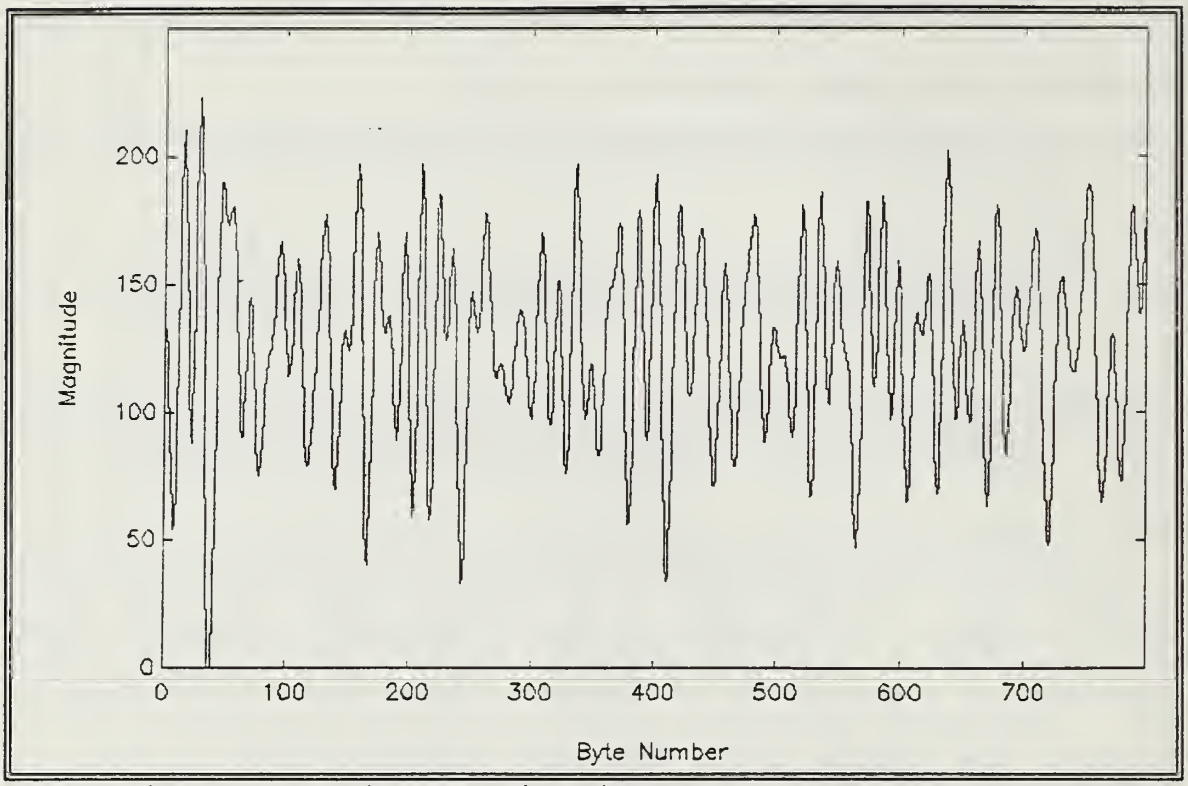


Figure 5. Time Domain Signal of a 256 Frequency Signal in a 4096 Bin System.

a whole and discusses the required hardware and software elements to make the system viable. The many elements of interaction and interdependence are discussed to enable the reader to develop a base of understanding for the choices made in the following chapters. Implementation options are discussed along with details of the experimental process in Chapter Three. The conclusions and recommendations for further study are the subjects of Chapter Four.

II. PC BASED MFQPSK COMMUNICATIONS

A. SYSTEM COMPONENTS

The major components in a personal computer based MFQPSK communications system are: (1) data input, (2) Fast Fourier Transform mechanism, (3) memory access system, and (4) digital to analog conversion.

The organization of these elements is coordinated by a main high level language program which calls subroutines to initiate the individual operations required.

B. DATA INPUT

The main program's first task is to retrieve the message character set and perform the frequency encoding. A subroutine would typically perform this task. In the final system implementation, it is supposed that the entire message to be encoded is readily available for translation. Messages are prepared independently of the communications system, but quite possibly with an editor on the same physical computer, with characters being represented in American Standard Code for Information Interchange (ASCII) format and placed together as a standard Microsoft Disk Operating System (MS-DOS) file on one of the computer's disk drives.

The encoding process must take care of several details to be used effectively: (1) It must confirm an appropriately-sized message, for example 4096 characters, fill the unused character area with a predetermined idle character if too short, or break the message into packets, each of which can be contained within the size constraints provided, if too long. If the message is broken into parts, the main program must have a means of automatically transmitting multiple segments. (2) The primary memory needed for the FFT subroutine parameters and scratch area must be dynamically allocated prior to the subroutine call and deallocated following. The exception to the deallocation is the case where several packets are being prepared for transmission as a single message. Allocated areas represented by a pointing vector would be maintained in a list of areas in main memory to be output. When the transmission started, each area would be transmitted in turn, with pointers representing that area being removed from the message's list of pointers and sent to the transmitting routine. (3) Appropriate phase and frequency information must be assigned based on the message given, retained and assigned the complex conjugate values in the corresponding symmetrically located frequency bins. (4) Finally, the encoding subroutine must initiate the FFT subroutine call if the FFT is to be completed in software or must pass the appropriate primary memory location to the main program if a hardware FFT is to be accomplished. If a multiple packet message was to be sent an FFT would be conducted repetitively by moving through the message list of pointers.

C. FAST FOURIER TRANSFORM

Once the process has the data encoded into frequency and phase information, the FFT must be performed. As mentioned in the previous section, this can be accomplished either in hardware or software.

In the hardware choice, the data must be passed from the computer primary memory to the expansion board where a provision will have to be made at this point for an off computer secondary storage method. A hardware FFT, to utilize its speed advantage over a software FFT, must position the entire array of data elements in such a way as to allow all data to enter the parallel FFT process simultaneously and then be able to output the data in a byte serial fashion for final digital to analog conversion.

In the software FFT case, the main program must now call the FFT routine. FFT routines typically are passed an address pointer to the data location and return the transformed data in the same primary memory space.

D. MEMORY ACCESS

Regardless of whether the data is transferred from the primary memory to the expansion board before or after the FFT is performed, a decision about the means by which it is transferred must be made, unless the computer's memory is to be paralleled with memory placed on the expansion board or the hardware device is to have some bus authority, possibly as a parallel processor. Paralleling memory has hazards which will be discussed a little in the next chapter on choices. In general, however, there are three reasonably routine methods of moving the data off computer: (1) high-level language transfer; (2) assembly-language transfer; and (3) **d**irect **m**emory **a**ccess (DMA). Each of these choices lead to different paths, each requiring further choices to be made. One major factor in the decision, common to all choices, is the computer's own housekeeping problems, specifically, the memory refresh requirement and the nature and means of executing instructions.

E. HOUSEKEEPING

1. MEMORY REFRESH

Computers use **d**ynamic **r**andom **a**ccess **m**emory (DRAM) for their main memory. DRAM consists of a bank of single transistor elements which are conveniently modelled by a bank of capacitors having either a charged or depleted state representing a logic one or zero respectively. As with all capacitors, the charge will tend to leak over time, with the less desirable devices having less internal capacitance, and therefore less ability to hold their charge for a specified time. These less desirable devices require recharge or refresh more often than the more desirable ones. A typical engineering judgement made by the memory device manufacturer sets the time between refresh cycles at 15.125 microseconds.

This refresh is carried out automatically by a single DMA channel. Since the priority of the refresh is higher than any other computer operation, including that of the main program execution, allowances or corrections must be made to prevent a negative impact on the program's performance.

2. INSTRUCTION FETCHING AND EXECUTION

Instructions are placed in a section of main memory along with the operands they may require for execution. The instructions have varying lengths and are fetched by the main processor one byte at a time. The rate for this instruction fetch is one byte every four clock cycles because it is basically a memory fetch. Therefore, the instructions which consist of several bytes may take many clock cycles to load into the processor before they can be executed.

One of the innovations of the Intel 8088 over earlier processors was the implementation of an advanced fetching mechanism where instructions can be fetched in advance of their being needed by the processor. A total of four instructions may be prepostioned like this. One difficulty in placing the instructions in the processor in advance is that if a program loop occurs or program control is passed to another location by a subroutine call, the processor is ready to execute instructions in the old path. In these cases, the instructions are flushed from the processor and the process is restarted. This time consuming process can actually slow down the execution of a program directly following a branch or loop.

Instructions are programs contained within the microprocessor. These little programs are called microcode. Execution times of these microcoded programs depend on the number of operations to be performed. Some instructions are fairly straightforward and consume few machine cycles. Examples of short instructions are register to register transfers or arithmetic rotates which typically require a single arithmetic logic unit cycle. The two instructions cited take only two clock cycles of the main processor to accomplish. Others such as the interrupt instruction are extremely complicated and perform many functions inside the processor. It manipulates the stack pointer six separate times, as well as manipulating a number of other registers and takes over 50 clock cycles to perform.

Regardless of the instruction or its length, the processor will complete the instruction in its entirety before passing bus control to another process.

3. HARDWARE INTERRUPTS

In addition to memory refresh, processor program control may be set aside by the occurrence of hardware interrupts. The three interrupts used by the computer are: Interrupt 0 which is invoked every time the Intel 8253 programmable interval timer count reaches zero, Interrupt 1 which is invoked every time a key is depressed on the system keyboard, and Interrupt 6 which reports a disk access complete.

These interrupts are maskable and can be disabled by selecting the appropriate mask at Port 21h (address of the interrupt controller).

4. SYSTEM IDLE

Any program controlled machine must either be held in a halted state or it must have a series of instructions to perform. The halted state is used whenever another processor has control of the bus as in the case of a DMA transfer. More often than not, however, some input is required from the keyboard for the operating system to perform its next function. The input required may be a data input for a program, or may be a command for the operating system itself invoking a program.

The program loop used by the system while waiting for input contains the two instructions at each end of the instruction execution time spectrum--a two clock cycle instruction and a 52 clock cycle instruction. This program loop keeps checking to see if any entry has been made to the keyboard. Since the time to type a key is so large compared to the time it takes to perform instructions, the keyboard interrupts the process and deposits the representation of the key in a buffer in memory and, if the key is the first to be placed in an empty buffer, sets a flag to indicate something in the buffer.

The read-only memory (ROM) basic input/output system (BIOS) contains the instructions which tell the computer how to perform this vital service of waiting for an input. The ROM-BIOS is proprietary and changes from machine to machine but in all machines the program loop looks something like the following (the numbers in parenthesis indicate the number of clock cycles required for the instruction to execute):

```
                mov ah,1     ;request service #1  (4)
    recheck:    int 16h      ;anything from keybd?(51)
                jz recheck   ;zero flag means no  (16)
                mov ah,0     ;request service #0  (4)
                int 16h      ;places char in ah,al(51)
                    .
                    .
```

Int 16h looks something like:

```
                cmp flag,1  ;true flag means keybd(9)
                            ;hardware interrupt # 1
                            ;has occurred--its
                            ;int service routine sets
                            ;the flag
                iret                                (32)
```

F. DIGITAL TO ANALOG CONVERSION

All implementations of this system perform the digital to analog (D/A) conversion on the expansion board off computer. Since the conversion is made continuously, some decision must be made regarding the method of latching the data and selecting the specific time to activate the latch. Once data is latched the conversion is made to an analog signal. The design must specify the form of the analog signal to be used. The three decisions required are: (1) the no signal voltage level; (2) the voltage maximum to minimum range; and (3) the level of accuracy chosen. One of the given parameters in this study was that an eight bit conversion was sufficiently accurate. This was a convenient choice since memory data is eight bits wide and the INTEL 8088 transfers eight bits at a time.

III. IMPLEMENTATION OPTIONS

A. DATA INPUT

Since the encoding process is currently unresolved and the subject of further research in another thesis project, a test harness called **ENCODEDATA** was written which allowed the selection of the signals for a test of the remainder of the system. **ENCODEDATA** is the first subroutine called in the main program which is located in Appendix A. The subroutine prompts the user to choose the phase of the selected frequency bin or makes provision for a random assignment of frequency phase information for all assigned frequency bins. It maintains a record of the choices made so that the phases of the symmetrically located frequencies are properly chosen to maintain a real signal output following the FFT.

B. FFT IMPLEMENTATION

One of the first choices made in the design process was whether to implement the FFT in hardware or software, because of the number of other system characteristics resting on the decision. Arguably there exists a third option, that the FFT could be calculated off the computer with the results for transmission submitted as a file. If such a choice were made, however, the main reason for having a personal computer-based system, the ability to have a completely integrated system, would be violated.

The concept design provided at the start of this project assumed that the transformation of the data to be transmitted would be received and encoded by an offline FFT being provided to the computer simply as a file of bytes to be transmitted. At the time this author joined the research project, the FFT was being accomplished in the **A** **P**rogramming **L**anguage (APL) language off-line with the resulting data file being transferred to the output device which was a simple barrel shift register feeding a D/A converter.

The decision was made to incorporate the FFT into the computer system to allow a completely integrated system. A hardware FFT implementation was rejected because of the current lack of a single chip 4096 point FFT device. A few notes on the current work in this area are included in the conclusions and recommendations area.

Following the decision to use a software FFT implementation, the algorithm and computer language needed to be chosen. As was mentioned earlier, the preliminary research used APL which is an interpretive rather than a compiled language. The author chose to use a much faster compiled language such as Pascal, C or Fortran for the final implementation.

One benefit of using a compiled program in place of an interpreted program is that, for an interpreted language, much more memory is required to be in use to support the program. Interpreted languages must keep every command structure available for use. A compiled program typically picks only those function categories required for the specific application to be loaded into memory for the program to execute successfully. For example, if screen output is not required, all routines concerning screen interface may be omitted from the compiled version of the program with no ill effect.

The biggest effect, though, is speed. The author found a standard Turbo Pascal FFT routine would run at about one-half of the time cost of using APL.[1]

The decision was made to use an FFT routine from a standard library. The FFT subroutine used for the test harness is called **ComplexFFT** and is part of the file **FFT87B2.INC** available as part of the Borland Turbo Pascal Numerical Methods Toolbox.[2] The Cooley-Tukey Algorithm is used in this implementation and requires approximately 22 seconds for a 4096 point FFT on an 8.0 MHz zero wait state machine with an Intel 80287 math coprocessor. The FFT is performed prior to the beginning of the message transmission.

---

[1]The actual times were 22 seconds for the Turbo Pascal versus 57 for the APL using the same machine with the most efficient algorithms known.

[2]Source code for this subroutine is not included because the license will not permit reprinting the source code. A program written by the license holder is permitted to include a compiled copy of the subroutine.

As with many FFT routines, the transformed signal is returned to the calling program in the same matrix (this also means the same primary memory locations) in which the input frequency and phase information were passed.

Flexibility was the factor which the author used to decide on the specific compiled language with all other elements being equal. FFT subroutines using the same basic algorithm were readily available in Fortran and Pascal, and could have easily been translated into C. Pascal was chosen over Fortran because of the ease in handling input and output and because of the fact that, since its entry (particularly the entry of **TURBO PASCAL** by Borland International, Inc. which dramatically changed the pricing structure for programming languages) into the personal computer arena, a wealth of standard routines have become available. All software needed for the development of this project had a cumulative cost of approximately $200 at commercial retail rates.[3]

[3]This cost included the Turbo Pascal Numerical Methods Toolbox, of which the FFT and related routines are a part, the Turbo Assembler and the Turbo Debugger, which were used for the Assembly language subroutines for initialization and startup of the DMA process, and Turbo Pascal, which was used for the main program home for the system. The cost of the operating system (approximately $100) is not included in these estimates.

C. DATA OUTPUT TECHNIQUES

Given that the FFT was to be performed in software, the next step in the process was to decide on the method of getting the FFT results from internal memory to the D/A converter. The largest problem initially considered in this area was that of accommodating the memory refresh overhead for the primary memory. Several approaches were examined: (1) postpone the refresh such that the entire block of data (4096 bytes) could be transmitted in between refreshes; (2) replace the dynamic memory with static memory which requires no refreshing; (3) use off computer static or bubble memory, loading the memory off computer in parallel with the on computer memory, and then isolating the computer bus during the transmission of the output signal; (4) use off computer static or bubble memory, transferring data to the off computer memory and then transmitting; and (5) use DMA in the single byte mode so that memory refresh could be accomplished in a normal manner.

The options were considered in much the order listed above. The following sections detail the process through which the decision was made.

1. POSTPONING MEMORY REFRESH

As mentioned in Section II.E, the frequency of this refresh depends on the capacitance of the poorest quality device in the system's memory. The computer uses DMA channel zero to perform the refresh, one block at a time, initiated by a pulse from the system timer (INTEL 8253). Each DMA channel has a priority assigned based on the number of the channel, with the lowest number, channel zero, having the highest priority. Refresh excludes the processor from the system bus for the period of refresh. The DMA controller keeps track of the address of the memory block to be refreshed next so that a new block is chosen each cycle. The cycle takes five computer clock cycles to complete.

The refresh time was compared with the most straightforward means of outputting data from the primary memory--an assembly language program loop. Appendix B contains a simple nine line loop program (Eggebrecht, 1983, p. 188) which was used in this test. A total of 43 clock cycles are required in the case of an INTEL 8088 processor to accomplish one pass through the loop's instructions. A test circuit was designed and used to verify the output timing. This operational test of the first circuit design also provided the first look at the effect of the memory refresh on a controlled output. The test waveform was extremely jittery.

The reason for the jitter was that while the loop consisted of 43 clock cycles, the main processor turned control of the busses over to the DMA channel zero memory refresh every 121 cycles for a period of five clock cycles.[4] The net result of this periodicity conflict was that sometimes two data bytes and other times three data bytes would be output between refreshes.

The first attempt to correct the jitter resulting from the memory refresh problem was to manage its timing to suit the system needs. There is a wide range of timings permitted for the memory refresh.

As mentioned previously, the memory manufacturers specify a typical refresh requirement of 15.125 microseconds. This is not a hard and fast rule, however. A recent article in the **PC Magazine** (Roemmele, 1988, pp. 331-346) discussed a method of manipulating the memory refresh period. Experimentation on Naval Postgraduate School machines yielded an experimental result that the refresh cycle could be extended to 30 milliseconds for all machines with some select machines being extended up to 0.3 seconds without parity error. The **PC Magazine** article cited an example of a machine which could be refreshed as infrequently as 1.001 seconds! The assembly language program cited in the article and used in this test is included in Appendix C.

[4]The times listed here are for the 8.0 MHz personal computer using an Intel 80286 processor used in this project. Refresh is based on time, not instruction cycles, whereas instructions are just the opposite.

Since the refresh period could not be expanded to include an entire block transmission in this mode, the refresh period was reduced to match one byte of data transfer. Every cycle of the program loop which output a single byte was followed by a memory refresh.

The real difficulty of using the program loop was its severe inflexibility. The entire communication system must be designed for a specific class of machines running at a specific frequency. If for some reason the output frequency had to be changed, the refresh period would have to be changed and the program loop would also have to be modified to include more or less no operation (NOP) statements. The range of output frequencies available under this method for an 8.0 MHz machine was limited to a low of approximately 66 KHz because of the memory refresh requirement and a high of approximately 145 KHz because of the time in executing statements in the program loop.

The first design, although technically meeting the requirements of the system, clearly needed more flexibility.

2. PRIMARY MEMORY REPLACEMENT

If static ram (SRAM) were used for primary memory, memory refresh would not be an issue. SRAM is a much more complicated structure which uses, instead of a single transistor holding a charge, eight transistors forming a flip-flop. This typically requires much more chip real estate and, as a result, costs much more per byte than the DRAM. The obvious next question was, "Why not just replace the DRAM with SRAM and not worry about the refresh?" Unfortunately, the two different types of chips are not interchangeable. A SRAM chip manufactured with the same technology as a DRAM chip will have less memory in the same package size. SRAM built under newer technology allows more compact placement of the circuitry and has a difference in the pinout which prevents a direct replacement. One option in dealing with this problem was to build a memory expansion board with SRAM and disable the old memory and refresh buses. This option was not attempted because of the drastic nature of the solution. It should be noted, however, that the frequency range under this option would lose its lower end limitation.

3. ADDING EXTERNAL COMPUTER MEMORY

Many times a simple conceptual design becomes greatly burdened when it gets to the point of real world implementation. The idea of paralleling memory is one of those things that sounds like a simple solution to the problem of transferring data but ends up being very complicated. The concept is that corresponding memories on the buses would be parallelled for the calculation part of the program and then isolated by means of a latched address decode following the FFT calculation. The expansion address and data buses would have to be isolable from those of the main processor. To meet the requirement that memory refresh would not affect the output, an entire second system of output address program control would have to be implemented to select the particular byte to be output at any given time during the second phase of the process where the busses were isolated. A 4096 byte barrel shifter was found[5] which could handle the output problem of the second phase reasonably well without a separate program counter, but had no means for being randomly addressed by the processor during the first phase. Many suitable random access devices were found which could meet the first phase requirements but had no reasonable way of producing the output independent of the processor's program control.

In addition to the complicated construction required, flexibility is again of concern. Only one memory region is allowed for the placement of data coming from the FFT. Any other program using the computer has to be concerned about the meaning of the latched decode address for isolating the expansion board and its conceivable initiation of the output process.

The research path that this series of problems seemed to propose next was to duplicate the memory in question without parallelling. This would allow the main processor to execute program control for loading the external barrel

[5]The IDT7M204 chip was manufactured starting in 1985 by the Integrated Device Technology Company, Incorporated. It uses a nine bit wide data first in first out (FIFO) array of 4096 bytes.

shifter and then pass control for the output directly to an oscillator which would gate the output on the expansion board. The advantages of this modification were to permit the computer housekeeping to proceed without any interference and to allow the output process to operate with only the limitations of the output device itself. As an example, the IDT7M204 device would allow a range of output frequencies between 0.1 Hz to 12.0 MHz.

The disadvantage was that the design still required a means of transferring the data from internal to external memory and imposed a limitation on the length of a continuously transmitted message to the length of the available external buffer--in this case only one block or packet of data.

Still, this approach was the solution pretty well settled upon until the problem of memory to memory transfer of data was taken on in earnest. It seemed reasonable that DMA was in order because raw speed was the prime concern in getting the data to the output device. However, once the study of the various DMA options was started, it became apparent that a DMA technique might well hold the answer for a direct output to the converter without the bother and inflexibility of an intermediate memory device.

Additionally, if several packets were to be transmitted in a single message, the transmission would have to stop while the intermediate device was reloaded from main memory with the next packet of information. The DMA technique would require only changing the base pointer in the DMA controller to effect the location of the subsequent data for output.

4. DMA OUTPUT

The DMA controller (Intel 8237) has two general modes of operation--block and byte. When performing a transfer in the block mode, no maskable interrupts or other coprocesses are allowed to take control of the bus. This can result in a failure of the refresh if the blocks are too large. The computer manufacturers specify this mode as unallowed for that reason, even though technically the block mode can certainly be programmed. When a block transfer is initiated, the computer automatically transfers the block at the maximum DMA rate which varies among the various computer manufacturers. This transfer takes six clocks because of a wait state inserted by the baseband logic to accommodate slower memory (Eggebrecht, 1983, p. 115). The hardware designer also has it within his ability to insert additional wait states if needed to slow the process down. For a block of 4096 bytes and an 8 MHz machine, this could fall within the bounds of the extended memory refresh. If this method were to be used for the system 4096 byte block, the transfer could occur at an incredible byte rate of approximately 1.3 MHz taking a little over three milliseconds to transfer the entire block! Unfortunately, if transmission is desired at a slower rate, the issue of flexibility again raises its head. Since the 4096 byte transfer pushes the system to its practical refresh limits, any solution requiring additional wait states to be inserted extends the cycle beyond acceptable limits and will certainly result in parity error failures of the main memory.

This mode was considered with a simple modification to the system of halving the size of the block of data. This approach allowed the data to be transferred at any rate between 650 KHz and 1.3 MHz by placing the I/O CH RDY line in the inactive low state while waiting for the next clock. By reducing the block size to half, the refresh requirements continued to be met. The difficulty in this approach had to do with the specific system being implemented which needed a wide number of frequency bins available in the lower frequency spectrum. As the frequency of byte transfer increased, the size of the frequency bins increased as well. This, coupled with the restriction that the block size be cut in half, drastically reduced the number of frequency bins available in a given low frequency response region. For example, a frequency response of 50 Hz to 15 KHz (such as a normal acoustic channel), sending information with a 2048 bin system at a sampling rate of 650 KHz, has a bin size of approximately 635 Hz and only 23 frequency bins available for assignment in the permitted frequency range. Contrast this with the 4096 bin system sampled at 60 KHz which has a bin size of approximately 29 Hz and 515 bins available for encoding within the specified frequency response.

The DMA single byte transfer mode, even though it takes a few more clock cycles to perform, offered more flexibility in the design. The flexibility comes from the ability to vary the frequency over a wide range of values by requesting a byte transfer almost at will. Each byte is output after an active request has been asserted on the appropriate DMA request line.

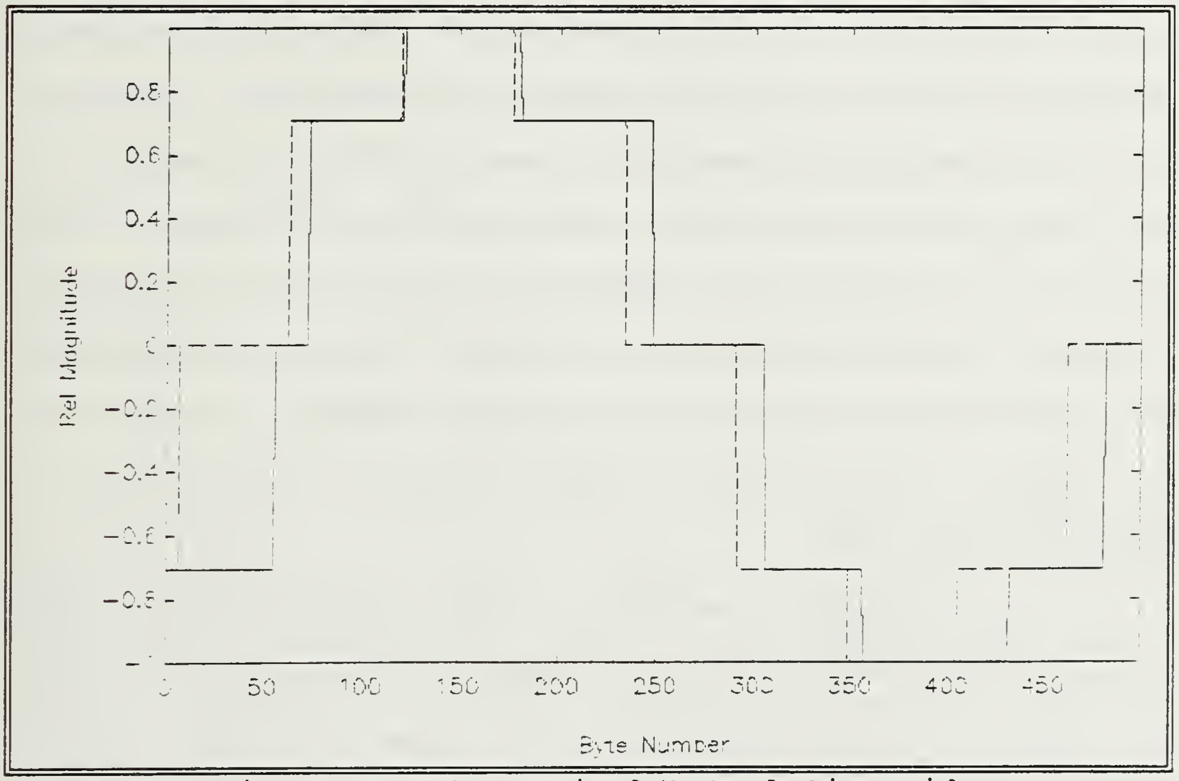


Figure 6. Theoretical/Actual Sinusoid

When the choice was made to implement this design, it was expected that the range of allowable frequencies would be from 0.1 Hz to 880 KHz. This derived from the fact that, in this mode, the main processor is allowed an instruction cycle (four clock cycles) in between every DMA byte transfer (six clock cycles). Thus only about half of the instruction bandwidth is utilized for the DMA transfer. Unfortunately, the background process detailed in Section II.E.2 consisted of instructions much longer than the four clock memory access cycle. Given the longest instruction of 51 clock cycles (INT) and the six clock DMA cycle, the highest sampling frequency with a 100 per cent probability of noninterference was 140.35 KHz. Even given the fact that the signal loses no pulses, the signal is not truly stable. Because of the differing lengths of instructions in the system idle loop, the latched signal is transmitted at times which fluctuate slightly around the DMA request. This effect is seen in Figure 6 and is not a significant problem.

An incomplete summary of the events causing the sinusoid perturbation shown in Figure 6 is listed below with the timing references signifying the number of elapsed clock cycles:

Cycle	**Event**
0	INT instruction occurs and requires 51 clock cycles to complete.
1	DMA requests the bus for byte transfer.

50	DMA cycle starts.
57	CMP instruction occurs and requires 9 clock cycles to complete.
57+	DMA requests the bus for byte transfer #2.
67	DMA cycle #2 starts.
74	IRET instruction starts and requires 32 clock cycles to complete.
107	JZERO instruction starts which flushes the instruction pipeline but causes no problem since it requires enough time for execution to allow pipeline to refill. It requires 16 clocks to complete.
114	DMA request #3.
116	DMA cycle #3 starts.
123	INT instruction starts (51 clocks).
171	DMA request #4.
175	DMA cycle #4 starts.
182	CMP instruction starts (9 clocks).
192	IRET instruction starts (32 clocks).

Note that the very worst case is the starting point for the process, yet, by the second data point the signal is stable at an optimum value within a very narrow time differential. Since there are two instructions in the loop with substantially shorter instruction lengths than the troublesome INT instruction, there is a significant cushion available to the process to buffer the effects of the different execution lengths.

When viewed on the oscilloscope the output appears stable up to approximately 250 KHz. This is due to the fact that the timing of the computer is completely independent of the timing of the crystal oscillator for the expansion board allowing the instruction performance to be much more well behaved than the worst case cited above where the longest possible instruction occurs just the instant before the clock pulse requesting a byte transfer. Depending on the level of redundancy in the code, the errors introduced by operating in the range between 140 KHz and 250 KHz may be acceptable.

Another interesting possibility arises when the output is accomplished at the lower rates. Less than two percent of the instruction bandwidth of the computer is utilized at 140 KHz. This leaves most of the computer's capabilities free to accomplish other tasks in parallel with the message transmission.

The memory refresh is easily accommodated in this method as well since a window is opened every byte transfer cycle which permits another higher priority process such as the refresh to take place.

The DMA must be initiallized when its use is to be invoked. Initialization is accomplished by writing data to several registers contained within the DMA controller. In the personal computer design, the address space starting at Address 0 is decoded and sent to the DMA controller. This application uses an assembly language routine disguised as a Pascal subroutine named **DMAINIT** (for **DMA init**ialize). This subroutine, which is located in Appendix D, initializes the base address of the matrix, the number of points to be output, the byte type of memory transfer, and the channel to be utilized. It was written closely following a non-working version used as an example in (Sargent and Shoemaker, 1984, p. 246).

Since the DMA controller is unable to address the entire address space of the computer, the DMA page register, a device separate from the controller, is initialized in the same subroutine with the source matrix address information. The address which decodes to this device is 80 hexadecimal.

DMAINIT is called as the last substantive step of the main program immediately following the **SCALEDATA** subroutine which ensures that an appropriate output level is obtained. The parameter which determines if a single block is to be repetitively output would probably be chosen differently in the system once a real encoded signal is being output. This parameter is a part of the word output to location dma+11. In this case, the DMA was initialized to automatically return to the same data block when the terminal count of bytes transferred was reached so that a nonchanging signal could be observed on an oscilloscope.

The current expansion board design has a DMA request occurring on the appropriate channel at a frequency determined by the oscillator input. This repetitive request is the result of an asynchronous clock running to the DREQ3* line. (The * symbol following the signal name indicates that it is an active low signal--in this case the request is being made when the signal is a logic zero.) As long as the channel is masked, the request has no effect and the output circuit ignores anything appearing on the data bus.

When the initialization is complete, the appropriate DMA channel is unmasked and the data transfer is started.

D. DIGITAL TO ANALOG CONVERSIONS

An octal data latch held the signal stable so that the conversion process from a digital to an analog signal could take place.

As can be seen in Figure 7, the data is stable (shown with a cross-hatch) on the data bus when the IOW* signal transitions from the active low state to the inactive high state in conjunction with the DACK3* signal being in its active low state. Coupling the logical or with the rising edge sensitive flip-flop a single point is defined for the

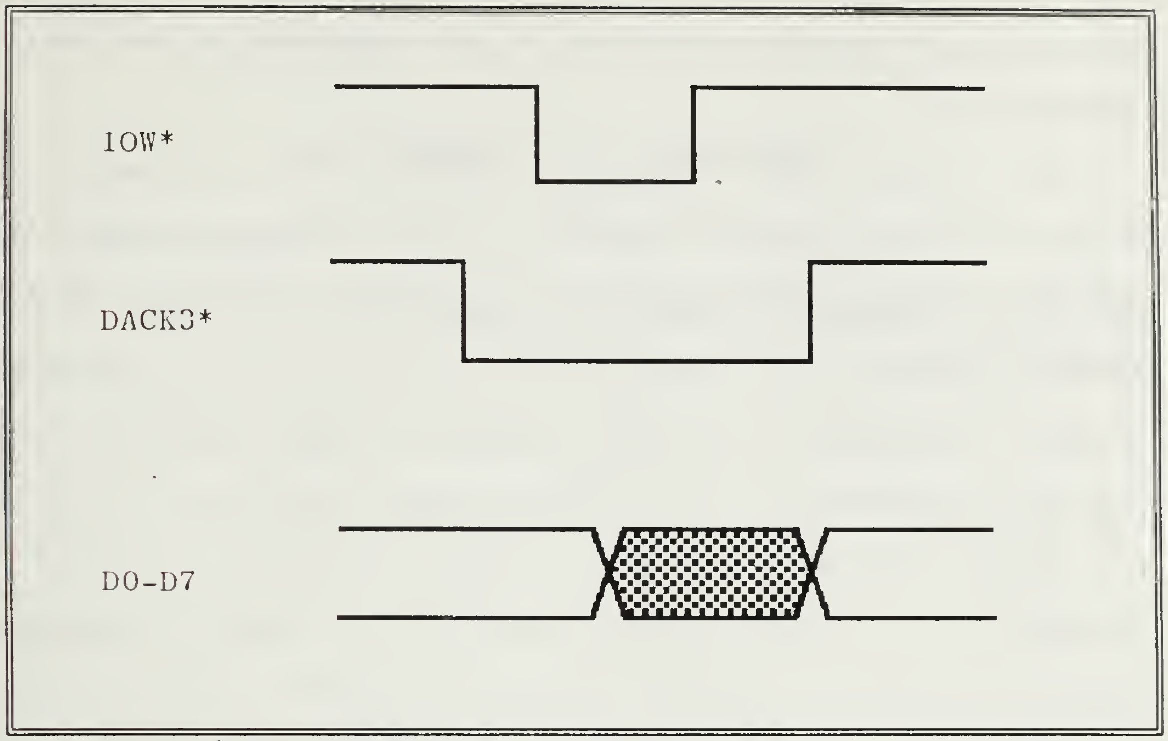


Figure 7. DMA Initiated Read From Memory

data latch. This is the point that primary memory is latched to retain the value until the next valid data.[6]

When a standard eight bit digital to analog converter (DAC 0800) was checked to meet the system specifications, its range of abilities was far beyond anything that the system needed both in terms of frequency and accuracy. The output of the octal latch is continuously fed to the converter where its

[6]It is necessary to include the DACK3* signal with the IOW* signal to prevent data from being latched whenever the IOW* signal becomes active due to a memory refresh. The logical or is chosen only because of part and signal availability. The two needed signals are available on the expansion bus only in the active low states.

output is provided to the back of the computer as the analog signal out.

The circuit was built on a Jameco JE36 PCB Breadboard which could be inserted directly in the expansion slot with output signals sent directly to a 25 pin connector on the computer back.

IV. SUMMARY AND CONCLUSIONS

A. FLEXIBILITY

A working MFQPSK personal computer based system was developed which not only meets the design constraints of a 10 KHz to 14 KHz frequency response output at a sampling frequency of 61.440 KHz but also provides a large measure of flexibility in the following ways: (1) the expansion board and driving programs are able to be run on any of the industry standard fully compatible computers using processors ranging from the Intel 8088 to the Intel 80386 with no software or hardware changes required; (2) the frequency response of the system can be tuned to match different hardware constraints imposed merely by changing the values of the constants ALOW, BLOW, CLOW, DLOW, ELOW, EHI in the main program of Appendix A which adjust the frequency bins selected by the data encoding process to match the frequencies permitted; and (3) by changing only a single clock on the expansion board, the output sampling frequency can be modified from 0.1 Hz to 140 KHz on an 8.0 MHz machine (the upper frequency limit can be extended to 1.3 MHz in the block transfer mode with the following block size limitations).

Frequency Range	Mode	Max Block Size
0-140K	byte	no limit
80K-160K	block	256
160K-320K	block	512
320K-650K	block	1024
650K-1.3M	block	2048
1.3M	block	4096

B. CHANGES REQUIRED FOR LONGER MESSAGES

With messages that result in a signal length longer than one block, **ENCODEDATA** should build a table of pointers representing the allocated memory locations where the blocks of data are represented. **ComplexFFT** should be called from within a loop that is executed until all blocks have been transformed. Since the FFT routine returns data in the same memory locations as when called, the table of pointers used on entry to the loop would be the same as those needed for output by the **SCALEDATA** routine. The **DMAINIT** routine would have to be modified to accept another parameter representing the current pointer to the base address. As mentioned previously, the autoinitialization option would not be chosen since each data structure would be transmitted only once with each call giving a new base address.

C. CURRENT AND FURTHER RESEARCH

As was mentioned earlier, current research is being conducted to implement a receiver for the system using a similar application of personal computer based principles. Another project is researching an error detecting/correcting scheme for translating the ASCII characters into the frequency and phase representations.

A project is needed to settle on a method for synchronizing the data blocks to protect against signal deterioration from the effects such as the transmission medium attenuation and differential channel delays for different frequencies. One intriguing concept worthy of further consideration is that a short correlation operation could be conducted as a parallel process utilizing the wasted portion of the instruction bandwidth mentioned in Section III.C.4 above. Since correlation requires only addition and subtraction operations which have a computational cost of approximately one-thirtieth that of the multiplies required for the FFT butterflies, the process could be designed to be conducted in real time during the transmission of a 4096 byte block of data. If not quite enough time was available for the synchronizing correlation, the block could be repeatedly transmitted the number of times necessary to extend the time of transmission while providing a simple means of error correction to the receiving system.

D. ENHANCEMENTS

The current progression towards very large scale integrated (VLSI) circuits makes it reasonable to expect an integrated device capable of performing a 4096 point FFT on a single chip within the next couple of years. A semicustom design could be pursued even now with a bitsliced approach from a company such as Advanced Micro Devices in Sunnyvale, California. With their AMD 29000 series devices, a user could design a special purpose data flow (meaning no program counter is required) microprocessor using standard library components such as floating point arithmetic processing units. Semicustom devices, however, are still very expensive and would not improve the system enough to justify the added expense.

Another potential approach which shows promise is to utilize a dedicated digital signal processor such as the products offered by the Zoran Corporation or Analog Devices, Inc., as a coprocessor. The trend with these devices is toward the capability of performing larger and larger FFTs on chip. The ZR34161 processor has an on device storage capability of 128 complex integers and can perform an FFT of that size in a single instruction. The continuing difficulty is that for each such FFT performed, 512 memory accesses must be made before and after the instruction is performed to preposition the data in the appropriate cache memory locations on the coprocessor. The promise held, however, is that when enough data can be prepositioned in the on chip cache memory, a real time FFT can be performed at a reasonable cost per application.

APPENDIX A

```
program THESIS;

{This program is the main program which supports the MFQPSK
system.  It is a Turbo Pascal program which runs on a MS-DOS
system with a math coprocessor.  Subroutines called by this
stub but not included in the file are ComplexFFT which is
contained in FFT87B2.INC file and DMAINIT which is contained
in the DMAINIT1.BIN file.  The assembly language source code
for the DMAINIT routine is contained in Appendix D of this
thesis.}

{$I-}                {Disables I/O error trapping}
{$R-}                {Disables range checking}

const
   TNArraySize = 4095;
                     {THIS IS THE SIZE OF A BLOCK}
   IOerr  :  boolean  =  false;

type
   NZERARRAY = array [1..256] of 0..4;
                     {FOR THE CHOICES OF ONE OF FOUR
                     PHASES OR FOR A NON-CHOICE OF
                     0.0 REAL AND 0.0 IMAGINARY}
   BCSTARRAY = array [0..TNArraySize] of byte;
                     {HOLDS DMA DATA TO BE BROADCAST.
                     THIS REPRESENTS AN ENTIRE BLOCK
                     OF FFT PROCESSED DATA WHICH IS
                     OUTPUT AS THE TIME SIGNAL}
   TNvector = array[0..TNArraySize] of Real;
                     {TYPE FOR BOTH THE REAL AND
                     IMAGINARY DATA FOR THE INPUT TO
                     THE FFT}
   TNvectorPtr = ^TNvector;
                     {PTR FOR FFT DATA ARRAY WHICH
                     ALLOWS DYNAMIC ALLOCATION OF
                     MEMORY BY THE "NEW" CONSTRUCT.
                     IF MORE ARRAYS WERE REQUIRED
                     TO BE KEPT FOR MULTIPLE BLOCK
                     TRANSMISSIONS, THIS DYNAMIC
                     ALLOCATION COULD KEEP THE DATA
                     AVAILABLE FOR MULTIPLE BLOCK
                     TRANSMISSIONS.}

var
   ALOW,BLOW,CLOW,DLOW,ELOW,EHI,I,INDEX,J,K,BINDEX,
      NUMBAUDS,NUMPTS,PHASECHOICE,TEMP  : INTEGER;
   DATAR,DATAI          : REAL;
   OUTFILE              : TEXT;
                    {THIS FILE WILL HOLD THE ASCII
                    CHARACTERS WHICH REPRESENT THE
                    HEX VALUES}
   THEFILE                   : file of byte;
                    {THIS FILE HOLDS THE ACTUAL HEX
                    BYTES}
   NZERO                     : NZERARRAY;
                    {KEEPS TRACK OF ASSIGNMENTS FOR
                    REVERSALS}
   BCST                      : BCSTARRAY;
                    {ARRAY MADE AVAIL TO DMAINIT}
   XREAL,XIMAG          : TNvectorPtr;
                    {PTRS FOR DATA SENT TO FFT}
   INVERSE, RANSELECT, SHORTCHOICE : BOOLEAN;
   DATA,ERROR           : BYTE;


{$I FFT87B2.INC}
{$I COMPFFT.INC}
{$I COMMON.INC}

procedure DMAINIT (var BCST : BCSTARRAY);
   {THIS PROCEDURE IS REALLY AN ASSEMBLY LANGUAGE ROUTINE
   BUILT IN THE FORMAT ACCEPTABLE TO TURBO PASCAL WHICH
   PASSES THE ADDRESS BCST ON THE STACK IN THE FORM OF FOUR
   BYTES OF DATA.  THE FOUR BYTES OF THE ADDRESS ARE
   COMPOSED OF THE SEGMENT (TWO BYTES) AND THE OFFSET (TWO
   BYTES).  THE PROCEDURE TAKES CARE OF ALL INITIALIZATION
   OF THE DMA CONTROLLER AND STARTS THE OUTPUT PROCESS.
   THE OUTPUT PROCESS COULD BE SEPARATED BY UNMASKING THE
   DMA CHANNEL SEPARATELY.  THIS PROCEDURE CAN BE CALLED
   REPEATEDLY BY MERELY SUBSTITUTING A NEW ADDRESS FOR A
   NEW 4096 BYTE BLOCK OF DATA.}
external 'DMAINIT1.BIN';
   {THE PROCEDURE CALLED HERE IS A BINARY FILE WHICH IS
   PRODUCED BY ASSEMBLING THE DMAINIT1.ASM PROGRAM, LINKING
   THE RESULTING DMAINIT1.OBJ FILE WITH THE MS-DOS PROGRAM
   LINK AND THEN EXE2BIN-ING THE FILE WITH THE MS-DOS
   PROGRAM EXE2BIN USING THE DEFAULT OUTPUT EXTENSION .BIN.
   THIS DMAINIT1.BIN FILE IS PLACED IN THE SAME DIRECTORY
   AS THE TURBO PASCAL COMPILER}

procedure GETPHASECHOICE;

   {THIS PROCEDURE IS WRITTEN ONLY TO BE ABLE TO SEE AN
   UNDERSTANDABLE RESULT OF THE FFT PROCESS WHICH IS A
   SIMPLE SINE-WAVE IF ONLY ONE FREQUENCY IS SELECTED.
   OF COURSE THE WHOLE PROCESS GETS MORE COMPLICATED IF
   MORE CHOICES ARE MADE BEFORE THE "NO MORE FREQUENCIES"
   OPTION IS SELECTED SINCE THE RESULT OF 256 SELECTIONS
   IS A BAND LIMITED WHITE NOISE CASE.  IT GETS THE PHASE
   CHOICES FROM THE KEYBOARD OR SELECTS THE PHASES RANDOMLY
   IF RANSELECT IS TRUE.}

var
   GOODRESPONSE : boolean;
   RESPONSE : char;

begin {GETPHASECHOICE}
   GOODRESPONSE := TRUE;
   repeat
      WRITELN('  WHICH ONE YOU WANT?');
      WRITELN;
      WRITELN('0       NOT THIS FREQUENCY, THANKS');
      WRITELN('1       QUADRANT ONE');
      WRITELN('2       QUADRANT TWO');
      WRITELN('3       QUADRANT THREE');
      WRITELN('4       QUADRANT FOUR');
      WRITELN('N       NO MORE FREQS PLEASE');
      WRITELN('R       DECIDE FOR ME, PLEASE (RANDOM)');
      READLN(RESPONSE);
      case RESPONSE of
         '0'            : PHASECHOICE := 4;
         '1'            : PHASECHOICE := 0;
         '2'            : PHASECHOICE := 3;
         '3'            : PHASECHOICE := 2;
         '4'            : PHASECHOICE := 1;
         'N','n'        :
              begin
                 SHORTCHOICE := TRUE;
                 PHASECHOICE := 4;
              end;
         'R','r'        : RANSELECT := TRUE;
      else
        begin
          WRITELN('YOU HAVE TO PICK ONE OFF THE LIST!');
          GOODRESPONSE := FALSE;
        end;
      end;{case}
   until (GOODRESPONSE or RANSELECT);
end{GETPHASECHOICE};

procedure ENCODEDATA

begin{ENCODEDATA}
         NEW(XREAL);       {THIS DYNAMICALLY ALLOCATES MEMORY
                            FOR THE ARRAY NEEDED TO INPUT
                            FREQUENCY BIN ASSIGNMENTS FOR THE
                            FFT ROUTINE}
         NEW(XIMAG);
```

```
FILLCHAR(XREALˆ,SIZEOF(XREALˆ),0);  {FILLCHAR FILLS
                  THE NEWLY ALLOCATED MEMORY WITH 0
                  VALUES}
FILLCHAR(XIMAGˆ,SIZEOF(XIMAGˆ),0);
ERROR := 0;      {THIS SETS THE ERROR RETURN CODE SO
                  THAT AN ERROR INDICATED BY THE FFT
                  ROUTINE CAN BE RECOGNIZED}
for I := 0 to NUMPTS-1 do
begin
   if (I < BLOW) then
   begin
      DATAR := 0.0;
      DATAI := 0.0;
   end
   else if (I >= BLOW) and (I < CLOW) then
   begin
      J := J+1;
      if not (RANSELECT or SHORTCHOICE) then
         GETPHASECHOICE
      else if SHORTCHOICE then
         PHASECHOICE := 4
      else
         PHASECHOICE := RANDOM(4);
      if (PHASECHOICE = 0) then
      begin
         DATAR := 80.0;
         DATAI := 80.0;
      end
      else if (PHASECHOICE = 1) then
      begin
         DATAR := 80.0;
         DATAI := -80.0;
      end
      else if (PHASECHOICE = 2) then
      begin
         DATAR := -80.0;
         DATAI := -80.0;
      end
      else if (PHASECHOICE = 3) then
      begin
         DATAR := -80.0;
         DATAI := 80.0;
      end
      else
      begin
         DATAR := 0.0;
         DATAI := 0.0;
      end;
      NZERO[J] := PHASECHOICE;
   end
   else if (I >= CLOW) and (I <= DLOW) then
```

```
          begin
             DATAR := 0.0;
             DATAI := 0.0;
          end
          else if (I > DLOW) and (I <= ELOW) then
          begin
             PHASECHOICE := NZERO[J];
             if (PHASECHOICE = 0) then
             begin
                DATAR := 80.0;
                DATAI := -80.0;
             end
             else if (PHASECHOICE = 1) then
             begin
                DATAR := 80.0;
                DATAI := 80.0;
             end
             else if (PHASECHOICE = 2) then
             begin
                DATAR := -80.0;
                DATAI := 80.0;
             end
             else if (PHASECHOICE = 3) then
             begin
                DATAR := -80.0;
                DATAI := -80.0;
             end
             else
             begin
                DATAR := 0.0;
                DATAI := 0.0;
             end;
             J := J-1;
          end
          else
          begin
             DATAR := 0.0;
             DATAI := 0.0;
          end;
          XREAL^[I] := DATAR;
          XIMAG^[I] := DATAI;
       end;
end{ENCODEDATA}

procedure{SCALEDATA}
begin
   WRITELN ('WRITING THE FFT RESULTS TO DISK');
   for INDEX := 0 to NUMPTS-1 do
   begin

   {*NOTE THAT THIS REPRESENTS A BIAS IN THE SAMPLE
   OF HALF THE DYNAMIC RANGE OF THE DIGITAL TO ANALOG
   CONVERTER.  THIS BIAS SHOULD BE SUBTRACTED BEFORE
   THE SIGNAL IS TRANSMITTED.  FOR EXAMPLE, IF THE
   DYNAMIC RANGE OF THE DEVICE IS 0-10V, THE BIAS IS
   +5.  THIS BIAS WAS DEVELOPED PRESUMING THE D/A DEVICE
   DOES NOT RANGE BETWEEN + AND - VALUES.}
      TEMP := ROUND(XREAL^[INDEX] + 126.0);
      if TEMP < 0 then
         TEMP := 0;
      DATA := TEMP;
      WRITE (OUTFILE,TEMP,' ');
      WRITE (THEFILE,DATA);
      BCST[BINDEX] := TEMP;
      BINDEX := BINDEX + 1;
   end;
end{SCALEDATA}

begin {MAIN PROGRAM}
   ASSIGN (OUTFILE,'OUTFILA1.DAT'); {THIS OUTPUT FILE IS
       DESIGNATED SO THAT AN EXPERIMENTER CAN ENSURE THAT
        HE HAS A REASONABLE OUTPUT PRESENTED TO THE DMA
        WHICH COULD CONCEIVABLY RESULT FROM THE FFT PROCESS
        TO WHICH HE FED THE INPUT.  IF HE SELECTED ONLY ONE
        FREQUENCY, HE WOULD EXPECT TO SEE IN THIS FILE THE
        ASCII REPRESENTATION OF A SINUSOID}
   REWRITE (OUTFILE);  {REWRITING MAKES THE FILE BLANK IF IT
        ALREADY EXISTS AND OTHERWISE CREATES A FILE BY THAT
        NAME}
   ASSIGN (THEFILE,'HOUTFA1.DAT');  {THIS OUTPUT FILE IS
        DESIGNATED SO THAT A BINARY OR HEXADECIMAL OUTPUT
        (DEPENDING ON YOUR POINT OF REFERENCE) IS CREATED
        WHICH CAN BE IMMEDIATELY OUTPUT TO A D/A CONVERTER.
        IN THE EARLIEST DAYS OF THIS PROJECT, THE DATA WAS
        OUTPUT THROUGH AN INDEPENDENT PROCESS FOR TEST AND
        THIS FILE WAS CONSTRUCTED SO THAT THE DATA WAS EASILY
        TRANSLATED}
   REWRITE (THEFILE);  {REWRITING MAKES THE FILE BLANK IF IT
        ALREADY EXISTS AND OTHERWISE CREATES A FILE BY THAT
        NAME}
   INVERSE := FALSE;   {THIS MEANS THAT WE ARE GOING TO ASK
        FOR A FORWARD FFT.  THE ISSUE OF WHETHER A FORWARD OR
        INVERSE FFT IS ONLY A MATTER OF A FACTOR OF 1/PI}
   J := 0;   {J PROVIDES THE INDEX FOR THE ARRAY WHICH
        RECEIVES THE 256 FREQUENCY BIN PHASE ASSIGNMENTS 0,
        1, 2, OR 3 REPRESENTING QUADRANT 1, 2, 3, OR 4, OR 4
        REPRESENTING THE CHOICE OF NO FREQUENCY ASSIGNMENT AND
        THEN RETURNS THAT VALUE IN THE REVERSE SEQUENCE FOR
        THE MIRROR FREQ ON THE OTHER SIDE OF THE NYQUIST
        CENTER FREQUENCY}
```

```
{THIS IS WHERE THE VALUES ARE INSERTED FOR THE VARIETY OF
CASES.  THE CATEGORIES ARE BROKEN UP AS FOLLOWS:  IN THE A
RANGE FROM ALOW TO AHI WILL ALWAYS BE ZERO REAL AND ZERO
IMAGINARY. IN THE B RANGE FROM BLOW TO BHI ARE A RANDOM SET
OF QUADRATURE PHASE SIGNALS.  THE B RANGE IS SELECTED BY
CORRELATING THE REQUIRED FREQUENCY RESPONSE OF THE INTENDED
OUTPUT DEVICE.  THE FREQUENCY PHASES ARE  SELECTED BY USING
A RANDOM NUMBER GENERATOR TO BUILD AN EVENLY DISTRIBUTED
SEQUENCE OF INTEGERS WITH VALUES RANGING FROM 0 TO 3 IF
RANSELECT IS TRUE.  IF SHORT CHOICE IS TRUE THEN THE VALUE IS
ALWAYS SELECTED AS INTEGER 4 WHICH REPRESENTS A NONCHOICE OF
FREQUENCY.  AS THESE VALUES ARE GENERATED THEY ARE STORED IN
THE NZERO MATRIX SO THEY CAN BE USED AGAIN FOR THE REVERSE
SEQUENCE NECESSARY TO GENERATE A REAL TRANSFORM.  AFTER EACH
VALUE IS GENERATED AND TEMPORARILY STORED IN THE NZERO MATRIX,
IT IS ASSIGNED A QUADRATURE PHASE REPRESENTATION BASED ON THE
INTEGER VALUE ASSIGNED.  I FOUND THAT A GOOD WEIGHTING TO GIVE
THESE QUADRATURE SIGNALS WAS 80.0 REAL AND 80.0 IMAGINARY
(WITH THE SIGNS DEPENDENT ON THE REPRESENTATION SPECIFIED BY
THE INTEGER VALUE).  THE C RANGE IS AGAIN ASSIGNED VALUES OF
ZERO REAL AND ZERO IMAGINARY AND CROSSES THE CENTER OF THE
FREQUENCY SPECTRUM BEING SENT TO THE FFT ROUTINE.  THE D RANGE
IS THE RANGE WHERE THE B VALUES ARE REVERSED IN ORDER AND
GIVEN THE VALUES OF THE B RANGE COMPLEX CONJUGATES.  THIS IS
ACCOMPLISHED BY READING THE NZERO MATRIX IN THE REVERSE ORDER
AND EVALUATING THE INTEGERS AS IN THE B RANGE EXCEPT THAT THE
IMAGINARY PARTS HAVE THE SIGNS REVERSED.  THE E RANGE FROM
ELOW TO EHI IS ZEROS IN BOTH REAL AND IMAGINARY PARTS.}


   NUMPTS := 4096;  {THE COMBINATION OF NUMPTS AND NUMBAUDS
       IS INTENDED TO ALWAYS END UP BEING A TOTAL OF 4096
       POINTS PRESENTED TO THE FFT ROUTINE.  FOR EXAMPLE
       ANOTHER SYSTEM COULD BE CHOSEN SUCH THAT NUMBAUDS MIGHT
       BE 16 AND NUMPTS BE 256}
   NUMBAUDS := 1;
   ALOW := 0;
   BLOW := 672;
   CLOW := 928;
   DLOW := 2768;
   ELOW := 3424;
   EHI  := 4095;
   BINDEX := 0;             {BROADCAST INDEX}
   RANSELECT := FALSE;      {IF TRUE SELECTS RANDOM PHASES}
   SHORTCHOICE := FALSE;   {IF TRUE SELECTS LESS THAN FULL LOAD
                            OF FREQUENCIES}
   for K := 1 to NUMBAUDS do
     begin
         ENCODEDATA
         WRITELN ('PERFORMING THE FFT');
         ComplexFFT (NUMPTS,INVERSE,XREAL,XIMAG,ERROR);
         WRITELN ('THE ERROR VALUE IS  ',ERROR);
         SCALEDATA
         DISPOSE (XREAL);
         DISPOSE (XIMAG);
     end;
   CLOSE (OUTFILE);
   CLOSE (THEFILE);
   DMAINIT(BCST);
end.
```

APPENDIX B

```
;
;   Assembly Language Output Loop Program
;
;   example was compiled with the DEBUG program
;   provided with all MS-DOS systems
;
;
XXXX:0100      mov dx,2000      ;puts the port address in
                                ;dx register
XXXX:0103      mov bx,0112      ;puts address of data area
                                ;in bx register
XXXX:0106      mov cx,2000      ;specifies 4096 words of
                                ;data to be output
XXXX:0109      mov al,[bx]      ;take word of data at address
                                ;pointed to by bx register
                                ;and put it in al register
XXXX:010B      out dx,al        ;puts byte of data in al on
                                ;the data bus
XXXX:010C      inc bx           ;add one to the buffer address
XXXX:010D      dec cx           ;decrease count remaining
XXXX:010E      jnz 0109         ;loop until all bytes are out
XXXX:0110      int 20           ;returns system control to DOS
XXXX:0112                       ;this is where the data is
;placed
```

APPENDIX C

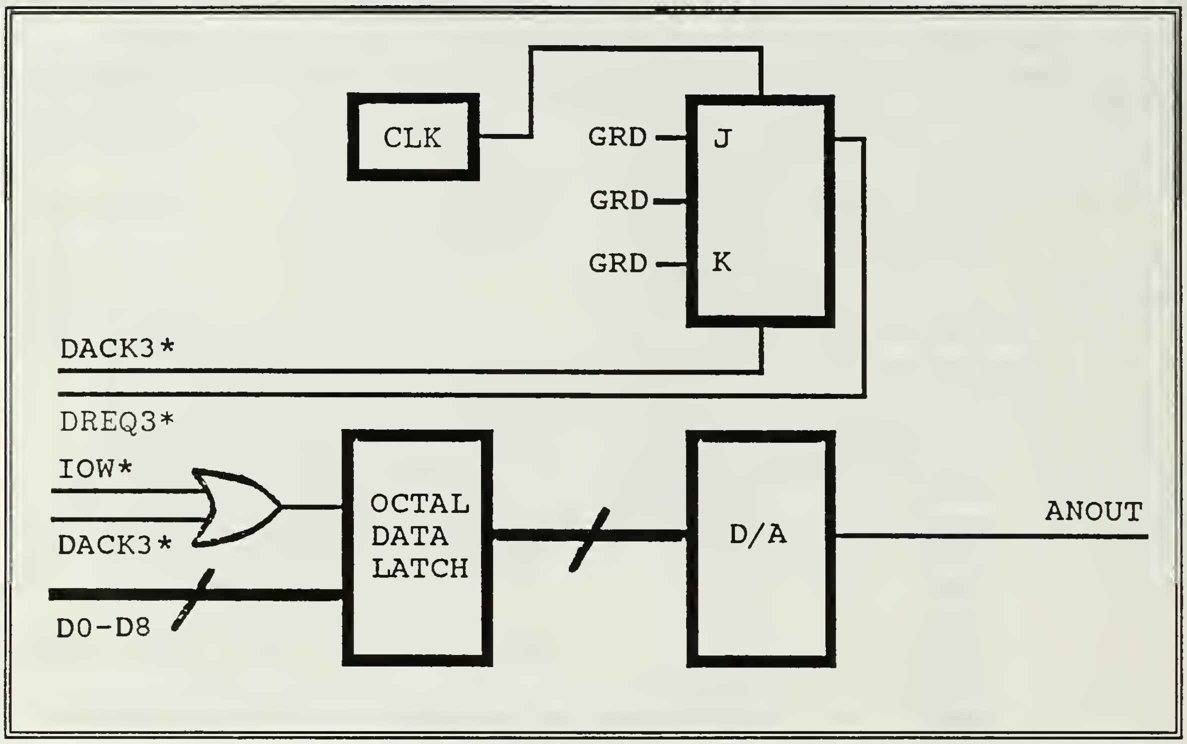
```
;
;   Assembly Language Program to Modify Refresh Time
;
;   example was compiled with the DEBUG program
;   provided with all MS-DOS systems
;
;
XXXX:0100     mov al,74       ;selects timer #1
XXXX:0102     out 43,al       ;43 is the port for the timer
XXXX:0104     mov al,ff       ;this is the value you will set
                              ;for the lower 8 bit count down
XXXX:0106     out 41,al       ;41 is the port for the count
                              ;preload
XXXX:0108     mov al,00       ;this is the upper 8 bits for the
                              ;count down
XXXX:010A     out 41,al       ;the timer keeps track of how
                              ;many times it has been sent a
                              ;value and knows that this is the
                              ;upper 8 bits since it is the
                              ;second time its address has been
                              ;decoded
XXXX:010C     int 20          ;returns system control to DOS
```

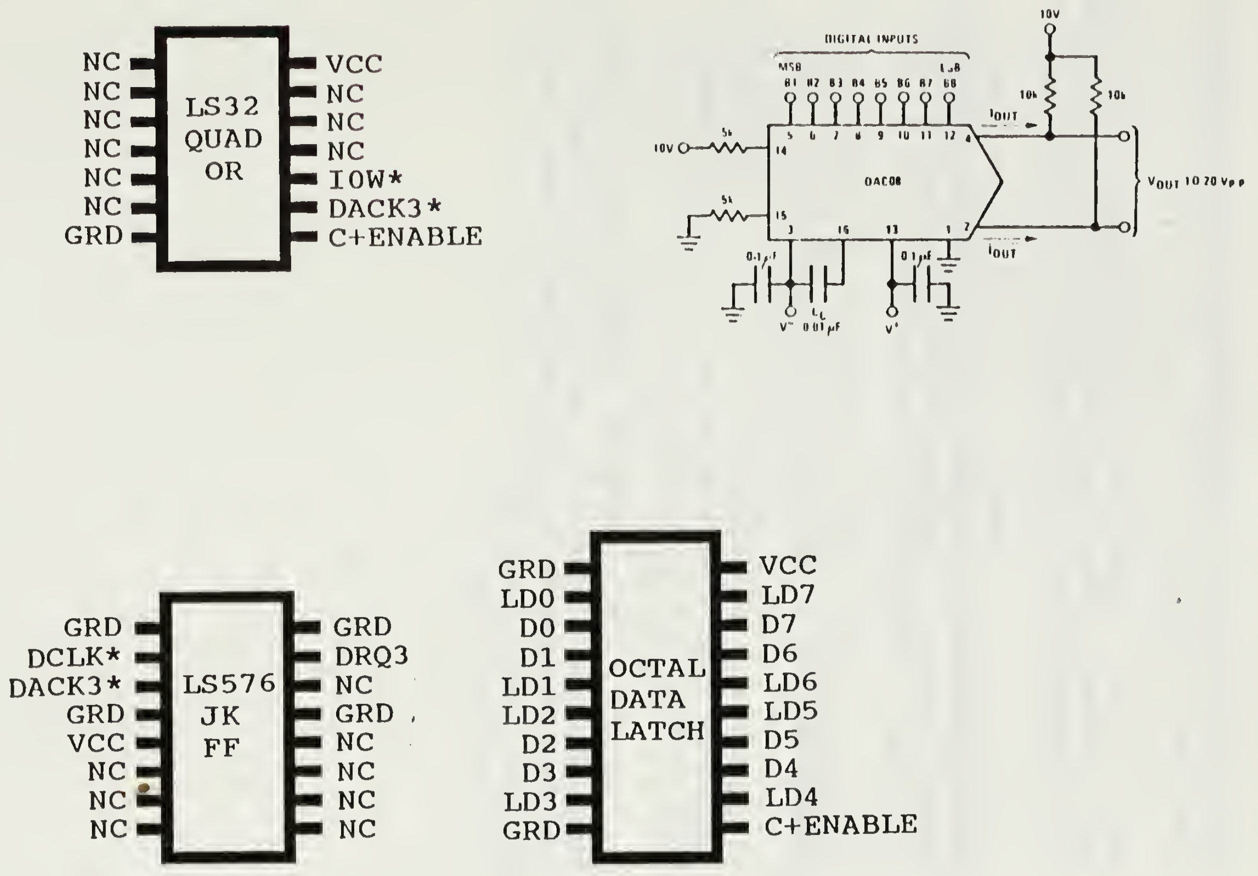
APPENDIX D

```
codeseg segment
    public dmainit
    assume cs:codeseg

;procedure DMAINIT (var BCST : BCSTARRAY);
;
;this procedure initializes dma channel 3 and sets the
;parameters to output the array bcst by passing the start
;address of the array on the stack.

           dma        equ         0
           dmapage    equ         80h
           dwavcnt    equ         1000h

dmainit    proc       near
           push       bp
           mov        bp,sp       ;use bp to address stack
           les        di,dword ptr[bp+4] ;move address of
                                  ;bcst into es:di
           mov        al,5bh      ;dma chan 3 single mode,
           out        dma+11,al   ;    read, autoinitialize
           out        dma+12,al   ;reset first/last ff
           mov        ax,es       ;calc high order 4 bits
           mov        cl,4        ;    of buffer area
           rol        ax,cl
           push       ax          ;save ax for dma start addr
           and        al,0fh
           out        dmapage+2,al ;store in ch 3 dma page
                                  ;register
           pop        ax
           and        al,0f0h
           add        ax,di       ;get page offset
           out        dma+6,al    ;output waveform buffer
           mov        al,ah       ;    start address
           out        dma+6,al
           mov        ax,dwavcnt ;output dma byte count
           out        dma+7,al
           mov        al,ah
           out        dma+7,al
           mov        al,3        ;unmask ch 3 to start
           out        dma+10,al
           pop        bp
           ret        4           ;pop 4 bytes off stack for
                                  ;addr of bcst
dmainit    endp
codeseg    ends
           end dmainit
```

APPENDIX E

The following two figures show the circuit used in the expansion board based MFQPSK system. The first is the logical schematic and the second is the actual expansion board pin diagram.





LS32 QUAD OR
NC NC NC NC NC NC GRD
VCC NC NC NC IOW* DACK3* C+ENABLE
DIGITAL INPUTS
MSB B1 B2 B3 B4 B5 B6 B7 B8 LSB
DAC08
10V
5k
10k
IOUT
VOUT 10 20 Vpp
0.1µF
0.01µF
V−
V+
LS576 JK FF
GRD DCLK* DACK3* GRD VCC NC NC NC
GRD DRQ3 NC GRD NC NC NC NC
OCTAL DATA LATCH
GRD LD0 D0 D1 LD1 LD2 D2 D3 LD3 GRD
VCC LD7 D7 D6 LD6 LD5 D5 D4 LD4 C+ENABLE

LIST OF REFERENCES

1. Eggebrecht, L. C., **Interfacing to the IBM Personal Computer**, Howard W. Sams & Co., 1983.

2. Gray, L. E., **Sampling Rate Reduction for a High Data Rate Acoustic Receiver**, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1986.

3. Proctor, E. L., **Design of a Digital Acoustic Communications Receiver for a Linear Hydrophone Array**, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1985.

4. Roemmele, B. K., "Instant Speedup for Your PC," **PC Magazine,** v. 7, no. 13, July 1988.

5. Sargent, M., and Shoemaker, R. L., **The IBM Personal Computer From the Inside Out**, Addision-Wesley Publishing Company, 1984.

6. Whitacre, P. M., **Effects of Soft Limiting on the Performance, Detection, and Synchronization of a Digital Acoustic Communications System**, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1986.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5100	1
4.	Professor P. H. Moose, Code 62 Me Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5100	6
5.	Commander Naval Ocean Systems Center Attn: Mr. Darrell Marsh (Code 624) San Diego, California 92152	1